CS 241, Homework #1, Fall 2009

Name: _____ (*and place your NetID on every sheet!*)

INSTRUCTIONS:

- This homework is **INDIVIDUAL WORK**. This homework **SHOULD NOT** be done with your MP team.
- Please **TYPE** your homework solutions. Handwritten solutions **WILL NOT** be graded.
- Please **STAPLE** all sheets together. And type your NetID on **EVERY** stapled sheet.
- Homework is due **IN-CLASS, AT THE BEGINNING OF CLASS (11:00am)** on Monday, October 12, 2009. Since solutions will be discussed during class, no late submissions are allowed.

GRADING RUBRIC:

	Multiple Choice /10	Function Call /6	Missing Code /4	Total /20
Grade:				
Grader:				

Multiple Choice Questions

The following C code compiles, but may contain some run-time erros:

```
char * foo(char *s)
{
  char *t = (char *)malloc(strlen(s) * sizeof(char));

  int i;
  for (i = 0; i < strlen(s); i++)
    if (s[i] >= 'A' && s[i] <= 'Z')
      strncat(t, s + i, 1);

  return t;
}
```

1. Which of the following is TRUE about the function foo()?
   a. The function does not always allocate enough heap memory to store the response string.
   b. The function relies on default value of allocated memory that may lead the function to access memory beyond the end of its allocated space.
   c. The function returns a reference to memory allocated to stack memory in the function's stack frame and cannot be read by the caller of the function.
   d. Both A and B
   e. Both A and C

CS 241, Homework #1, Fall 2009 NetID: _____

2. TRUE or FALSE: The system resources required to create and maintain a thread ARE GREATER THAN the system resources required to create and maintain a process.
    a. True
    b. False

3. Which of the following scheduling algorithms will have the LONGEST average response time after many jobs are queued and ran to completion?
    a. Round Robin with a quantum of much less than the shortest job
    b. Round Robin with a quantum of longer than the longest job
    c. Shortest Job First (SJF)
    d. Preemptive Shortest Job First (PSJF)
    e. First Come First Serve (FCFS)

4. TRUE or FALSE: Given enough binary mutexes, it IS POSSIBLE to create a counting semaphore but given enough counting semaphores it IS NOT POSSIBLE to create a binary mutex.
    a. True
    b. False

Consider the following code:

```
char course[] = "cs241";
printf("%d %d\n", strlen(course), sizeof(course));
```

5. What would be the output of the code provided above?
    a. 5 5
    b. 5 6
    c. 6 6
    d. 6 5
    e. None of the above

6. What would be the correct way to declare a pointer to the function: `char *myitoa(int)`?

    a. (char *) myptr(int);
    b. char *(my_ptr)(int);
    c. char (*my_ptr)(int);
    d. char *(*my_ptr)(int);
    e. (char *myptr)(int);

7. Which one of the following best describes POSIX call wait()?
    a. Pauses for a given amount of time.
    b. Waits for parent process to resume or issue an I/O call.
    c. Waits for any child process to complete.
    d. Waits for any child process to issue an I/O call.

8. Which of the following causes a POSIX **process** (not just the calling thread) to **exit**?
    a. A call to *pthread_exit* by a child thread but not the main thread.
    b. A call to *pthread_exit* by the main thread.
    c. A call to *pthread_join* by the main thread.
    d. A call to *return* by the main thread.
    e. A call to *return* by a child thread but not the main thread.
    f. Both D and E.
    g. None of the above.

9. The scheduler forced a process to switch from running to ready state. The scheduling algorithm must be:
    a. Non-preemptive
    b. Preemptive

10. The lifetime of a signal is the interval between the time of its generation and which one of the following events?
    a. Event of blocking the signal.
    b. Updating process signal mask.
    c. Event of expiration time for the blocked signal.
    d. Calling the corresponding signal handler for the delivered signal.
    e. sigwait() returns successfully after receiving the pending signal.
    f. D and E

Function Calls

In the following six questions, answer what SINGLE function or system call performs the operation described by the question.

11. Suspends a process waiting for a signal and returns # of received signal without calling handler

    Answer: _____

12. Creates a new POSIX thread

    Answer: _____

13. Sends a signal to a process

Answer: _____

14. Makes a copy of an existing process

Answer: _____

15. Returns process identifier

Answer: _____

16. Changes process priority

Answer: _____

Missing Code

A common task when manipulating strings in any programming language is tokenization. In fact, most programming languages, including C, provide a simple library call to do string tokenization. The following is the man page description of strtok():

```
char * strtok ( char * str, const char * delimiters );
Split string into tokens

A sequence of calls to this function splits str into
tokens, which are sequences of contiguous characters
separated by any of the characters that are part of
delimiters.

On a first call, the function expects a C string as
argument for str, whose first character is used as the
starting location to scan for tokens. In subsequent calls,
the function expects a null pointer and uses the position
right after the end of last token as the new starting
location for scanning.
```

(Question continues on the next page.)

The following small snippet of code shows strtok() is use:

```
char *input_string = "some string. to tokenize";

/*
 * On the first use, we provide strtok() the input_string
 * and strtok() returns a C-string. Notice that our
 * delimiter is the space character or the period.
 */
printf("%s\n", strtok(input_string, " .");
             /* Prints: some */

/*
 * On every use after the first use when we're tokenizing
 * the same string, we send in NULL as the first parameter.
 */
printf("%s\n", strtok(NULL, " .");
             /* Prints: string */

printf("%s\n", strtok(NULL, " .");
             /* Prints: to */
             /* Notice that both the period and the space
                are ignored from the input string. */

printf("%s\n", strtok(NULL, " .");
             /* Prints: tokenize */

printf("%p\n", strtok(NULL, " .");
             /* Prints: (nil) */
             /* Any call made to strtok() after the end
                of the string will simply return NULL. */
```

From the example code, you can see that subsequent calls to the first call of strtok() relies on information stored within the strtok() function. Specifically, we only supply the string we actually want to tokenize the first time we call the function. If multiple threads were to interweave their calls to strtok(), the result of the strtok() would be incorrect from thread to thread and would run differently every time the program is run based on when each thread ran its respective strtok() lines. Therefore, strtok() is not a reentrant function.

(Question continues on the next page.)

In this problem, you will write a reentrant strtok() function named strtok_r(). An outline of the code is provided below:

```
char *
strtok_r(char *string, const char *seps, char **context)
{
   char *head;  /* start of word */
   char *tail;  /* end of word */

   /* If we're starting up (first call to function),
      initialize context */
   /* --- LINE #1 --- */

   /* Get potential start of this next word */
   head = *context;
   if (head == NULL) return NULL;

   /* Skip any leading separators */
   /* --- LINE #2 --- */

   /* Did we hit the end? */
   if (*head == 0)
   {
      /* Nothing left */
      *context = NULL;
      return NULL;
   }

   /* Place tail at the next separator */
   tail = head;
   /* --- LINE #3 --- */

   /* Save head for next time in context */
   if (*tail == 0) { *context = NULL; }
   else
   {
      /* --- LINE #4 --- */
   }

   /* Return current word */
   return head;
}
```

17. Which of the following lines of code best fit into the comment space labeled "LINE #1"?

(A): if (string) { *context = string; }
(B): if (!string) { *context = string; }
(C): if (string) { head = string; }
(D): if (!string) { *head = string; }
(E): if (!string) { head = string; }

18. Which of the following lines of code best fit into the comment space labeled "LINE #2"?
    (A): while (!head) { head++; }
    (B): while (tail != head) { tail++; }
    (C): while (*head && strchr(seps, *head)) { head++; }
    (D): while (*head && *tail && strchr(seps, *head)) { head++; }

19. Which of the following lines of code best fit into the comment space labeled "LINE #3"?
    (A): while (*tail && !strchr(seps, *tail)) { tail++; }
    (B): while (!tail) { tail++; }
    (C): while (strchr(seps, *tail)) { seps++; }
    (D): while (!strchr(seps, *tail)) { tail++; }
    (E): while (*head && *tail && strchr(seps, *head)) { tail++; }

20. Which of the following lines of code best fit into the comment space labeled "LINE #4"?

    (A): *context = head;
    (B): *tail = '\0'; tail++; *context = tail;
    (C): while (*tail != 0) { tail++; }; *context = tail;
    (D): *context = tail;
    (E): while (*tail != 0) { tail++; }; *context = tail - 1;